

On steering coupled models.

P. V. Coveney, G. De Fabritiis, and M. J. Harvey

*Centre for Computational Science, Department of Chemistry, University College London,
Christopher Ingold Laboratories, 20 Gordon Street, London, WC1H 0AJ.*

S. M. Pickles and A. R. Porter

Manchester Computing, University of Manchester, Oxford Road, Manchester, M13 9PL.

Coupled models are set to become increasingly important in all aspects of science and engineering as a means to study complex systems in an integrative manner, particularly in systems biology and materials science. In computational terms, our aim is to embrace, within a single simulation, all necessary length and time scale processes to describe the system of interest. Such coupled, hybrid simulations typically communicate data between the component models of which they are comprised relatively infrequently, and so a Grid is expected to present an ideal architecture on which to run them. Computational steering is a powerful tool for interacting with simulations as they run, allowing the user to alter physical parameters, algorithmic parameters or even to migrate their simulation to another machine on the Grid. Hybrid models present new challenges in these respects. In the present paper, we describe a simple, flexible and extensible architecture for coupled models based on the RealityGrid computational framework and discuss its implementation for the case of a two-component hybrid molecular-continuum fluid dynamics scheme.

I. INTRODUCTION

Multi-scale modelling of physical systems is often employed to enable an accurate but computationally expensive model to be applied to regions of a system where it is required while a less accurate model can be used elsewhere. This enables far larger systems/longer timescales to be modelled than can be tackled using the accurate model alone.

Within the RealityGrid project (<http://www.realitygrid.org>) we are investigating the use of coupled models in two areas. The first of these is the process of nano-indentation where a nano-scale tip is pushed into a semiconductor surface and then retracted. The interaction of the tip with the surface must be modelled atomistically using quantum mechanics but this method is too expensive to be applied to the bulk of the surface. Instead, a continuum method is used to model the propagation over the surface of the stresses resulting from the indentation [1]. The second area of application is the study of fluid flow over a surface. Here, the interaction of fluid molecules with the surface is modelled atomistically using classical molecular dynamics while the bulk of the fluid flow is modelled using a continuum method.

A multi-scale model may be constructed by coupling together different simulation codes, each specializing in modelling a region of the system on one particular scale. Such a model then consists of a set of two or more codes which are synchronized through the (repeated) exchange of information as they execute. A coupled model could be written as a single executable code, but for flexibility, scalability and ultimately, performance reasons, we consider the case of a coupled model constructed from independent components which are interfaced to allow them to exchange information. These components are then free to be deployed in

such a way as to make the best use of available resources; something that is particularly important in a Grid environment.

Over the course of the RealityGrid project, the introduction of computational steering functionality to existing scientific codes has proved key [2, 3]. The process of “computational steering” refers to the interaction of a scientist with his/her running application. At its simplest, this may consist of monitoring the values of a few key variables to check the progress of the application. More advanced uses may involve the scientist altering variables that control algorithms within the code or altering parameters of the physical problem. Such use enables the scientist to further develop and apply their intuition to the investigation of the system under study. On-line visualization of the system is a powerful aid in this process since it can provide the user with visual feedback on the consequences of their steering activity.

When compared to steering multiple, independent executables, implementing computational steering for coupled models presents some additional challenges. In this paper we show how the RealityGrid steering system has been extended to handle coupled models and discuss a test case based on a two-component, molecular dynamics-hydrodynamics model.

The remainder of this paper is laid out as follows. In section II we describe the way in which we combine information from each component in a coupled model to produce a description of the coupled model as a whole and thus allow a steering client to treat it as a single, steerable application. In section II E we discuss the approach taken to avoid problems with deadlocking in our multi-service architecture. In section III we describe the application of this work to the case of hybrid molecular dynamics — a two-component coupled model

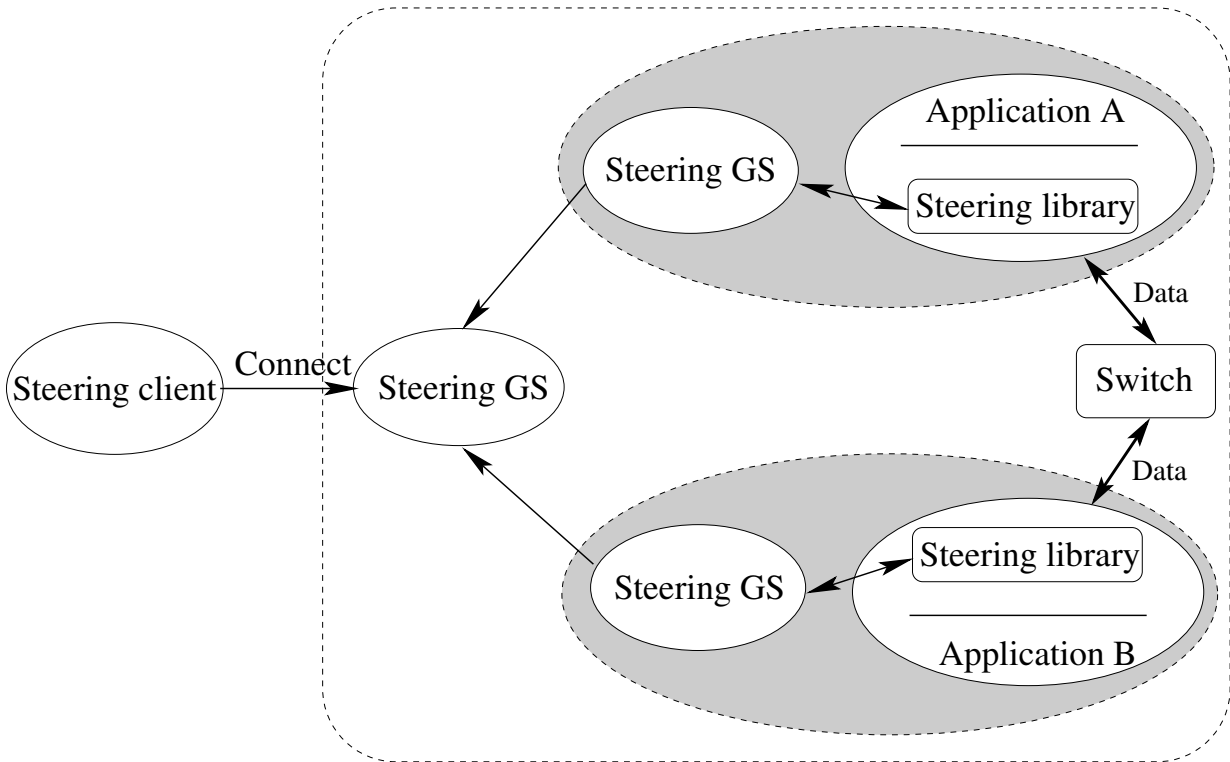


FIG. 1: The architecture of a steered coupled model consisting of two components with a separate “switch” (see section III B) to enable data transfer between them. The coupled model appears as a single component to the steering client.

for studying fluid dynamics. We describe the architecture of this system, the communication scheme used and the way in which synchronization of the components is achieved. Finally, in section IV we discuss the outlook for coupled models and the application of computational steering.

II. REPRESENTING MULTIPLE COMPONENTS AS A SINGLE APPLICATION

In this section we describe how the existing RealityGrid steering system has been extended to allow a multi-component coupled model to be steered as a single application.

Since a coupled model ultimately represents a single physical system, it is more intuitive and thus highly desirable for a user wishing to steer it to be able to treat it as such, rather than having to consider its constituent components. In addition, this approach has the advantage that existing steering clients, originally designed to connect to one or more independent components, can be used unchanged to steer a coupled model consisting of a number of components.

The RealityGrid project has developed a computational steering API and associated library [2, 4], designed to facilitate the introduction of steering into an application code. Given that a large number of applications within the project and an increasing number from outside are instrumented with this library, we aimed to design a system for

steering coupled models that would require no API changes.

In light of these requirements, the architecture of our system for steering coupled models is of the form shown in figure 1. As in the standard RealityGrid steering system [2], each individual component is represented by a Steering Grid Service (SGS) — a web service endowed with state and lifetime. In order for the coupled model to present a single interface to a steering client, we introduce a third SGS as shown in the figure. This SGS has responsibility for taking the information available to it from its children (the SGSs representing each component) and presenting it to the client in a suitable form. Thus, the vast majority of the extensions to the RealityGrid steering system to cope with coupled models are restricted to the SGS. Although figure 1 shows a coupled model with only two components, this approach generalises to more complex configurations, possibly requiring a deeper tree of SGSs or SGSs with more than two children (both of which are supported).

Within RealityGrid, each steerable component is characterised by certain metadata that is created by the steering library at runtime using the information supplied by the application. This metadata consists of:

1. the steering commands supported (*e.g.* ‘stop’, ‘pause’);
2. the set of monitored (read-only) parameters;
3. the set of steerable (writable) parameters;

```

<MSGSG:Coupling_config>
  <Global_param_list>
    <Global_param name="targetTemp">
      <Child_param id="http://machine.address:50000/MetaSGS/service/8312105489098098"
        label="8312105489098098/T_target"/>
      <Child_param id="http://machine.address:50000/MetaSGS/service/8312105489023033"
        label="8312105489023033/desiredTemp"/>
    </Global_param>
  </Global_param_list>
</MSGSG:Coupling_config>

```

FIG. 2: An example of metadata describing a single global parameter for a two-component system. The new, global parameter is given the name `targetTemp` and represents the parameter `T_target` from one child and `desiredTemp` from the other. The `id` attribute of each `Child_param` element identifies which child the original parameter belongs to.

4. the various input/output channels for data IO (*e.g.* for on-line visualization);
5. one or more checkpoint types for checkpoint control.

In order for a coupled model consisting of two or more components to present a single steering interface, each of these various aspects of each component must be combined in some way. In our architecture, this is performed by any SGS that has SGSs as its children. Thus, in figure 1, the left-most SGS is responsible for processing the metadata of the two SGSs which communicate directly with the two components of the application.

We now consider how each aspect of the component metadata belonging to each of the children of an SGS should be combined.

A. Steering commands

Since steering commands are generic to a steerable application, in order for a parent to support a command, it is necessary and sufficient that all of its children support it. Thus, the set of steering commands supported by a parent may be generated by AND'ing the sets of commands supported by each of its children.

B. Monitored parameters

The monitored parameters of a component are specific to it since they are a part of the state of that component. For instance, two components may both happen to have a monitored parameter representing the current temperature of the simulated system. Although these parameters represent the same physical quantity, they are very unlikely to have the same instantaneous value. In addition, they may provide useful information on the behaviour of the two separate components and the correctness or otherwise of the coupling between them. Consequently, the childrens' sets of monitored parameters are simply joined to form the set of monitored parameters of the parent (albeit with

tags added to their labels to ensure they remain distinct).

C. Steerable parameters

Steerable parameters are the most challenging part of a component's metadata to combine. This is because it is possible that steerable parameters belonging to different children actually represent the same physical quantity and thus must be represented by a single steerable parameter in the parent. To return to the temperature example of section II B, imagine that the same two components each have a steerable parameter controlling the *target* temperature of the system. (These parameters may well have different names.) Since both components are simulating (parts of) the same system, the value of this target temperature must be the same in both.

We describe such parameters as being *global* within the scope of the parent. Any change that a user makes to such a parameter must be passed down to the components in such a way that the change occurs in synchronized fashion, *i.e.* at equivalent points in simulated time.

Our system allows the user to construct metadata to describe such global parameters. An example is shown in figure 2. This metadata is supplied to the parent SGS which then uses it to perform the necessary processing of its childrens' parameters. (Since all messaging and metadata within the RealityGrid system uses XML, it is well suited to processing in this way.)

Once a parent SGS has used the supplied metadata to process its childrens' steered parameters, the set of steered parameters that it will supply to any attached steering client will contain the `targetTemp` parameter in place of both `T_target` and `desiredTemp`. Thus the steering client will have a single steerable parameter to control the target temperature in both of the components. Currently, our system assumes that the parameters making up a global parameter are all in the same units although any necessary scaling factors could, in principle, be included in the metadata.

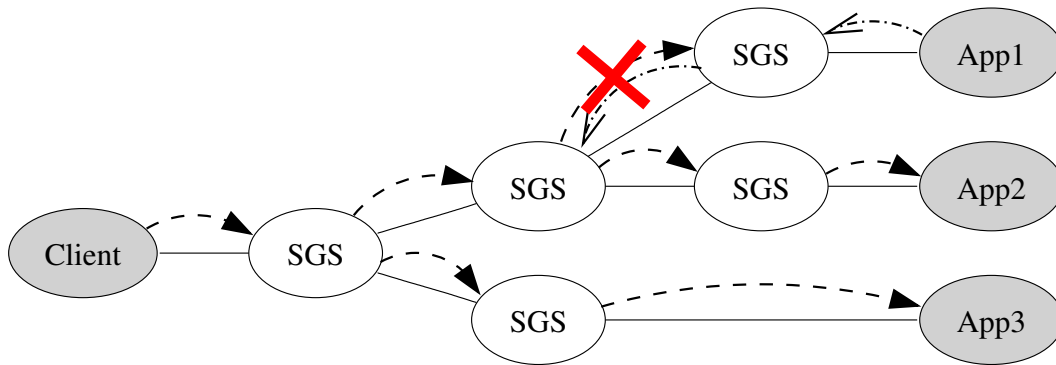


FIG. 3: An illustration of a potential deadlock scenario in a three-component coupled model. The component codes and steering client are indicated by filled ellipses and communication channels are shown as solid lines. In this figure the steering client has issued an instruction to the coupled model. This instruction must propagate down to each of the components, as shown by the dashed arrows. Simultaneously, component ‘App1’ has emitted a status message (dot-dashed arrows) which must propagate up the tree to eventually reach the client. Ultimately, this can result in deadlock as two SGSs make blocking calls to one another (indicated by the cross).

When the user steers a global steered parameter, it is the parent SGS’s responsibility to ensure that the change to the constituent parameters occurs at the same simulated time in each component. In order to achieve this, the parent SGS must have knowledge of the current simulated time of each of its child components. This is achieved by requiring each component of a coupled model to register a special steered parameter controlling the value of the time step. Given that the steering library can monitor the number of timesteps completed, it can then also calculate the total amount of time that has been simulated and make this available (as a monitored parameter).

Given knowledge of each child component’s time-step and current simulated time, the parent SGS can calculate a suitable future simulated time, t_{target} , at which to make the change to the constituents of the global parameter. It then inserts this information into the control message being sent to each child. The steering library has been extended to make use of this information and only apply the parameter change once the specified t_{target} time has been exceeded.

D. IO channels and Checkpoint types

As with monitored parameters, the IO channels associated with each component remain distinct. These may be used to connect suitable visualization packages to one or more of the components and thus need to retain their identity.

In contrast, checkpointing is a more complex issue. Provided that all of the components forming the coupled model have registered at least one checkpoint type then, in principle, it is possible to checkpoint the whole application. As with steering a global parameter, it is important that each of the components create their respective checkpoints at an equivalent point in simulated time. Therefore,

the same approach of generating a t_{target} and including it with the command to checkpoint sent to each component may be used.

E. Avoiding deadlock

Although the introduction of additional SGSs enables us to build a system capable of steering a multi-component application, it comes at the cost of some complexity. In particular, there is significantly more inter-service communication since information from a steering client must propagate down to the components being steered and vice versa.

We have adopted a model that hides the latency associated with reading messages that is introduced by the middle tier of SGSs. In our approach, any information that is *written* by a component is passed as far as possible through the tree of SGSs so that it is as close as possible to its destination. For instance, if the steering client in figure 3 wants to send a control message to the three-component coupled model then it *writes* that message to the SGS to which it is directly connected. That message is then passed from service to service (left to right in figure 3), undergoing processing as necessary, until it reaches the SGSs to which the application components are directly connected. It remains on each of those SGSs until the associated components *read* it from them.

Similarly, if one of the application components wants to send a status message to the steering client then it *writes* that message to the SGS to which it is directly connected. That message is passed from service to service, again undergoing processing as necessary, up the tree until it reaches the top-level SGS (the root of the tree) where it remains until the steering client *reads* it.

Since our SGSs are single-threaded, they can only service one call at a time. As illustrated

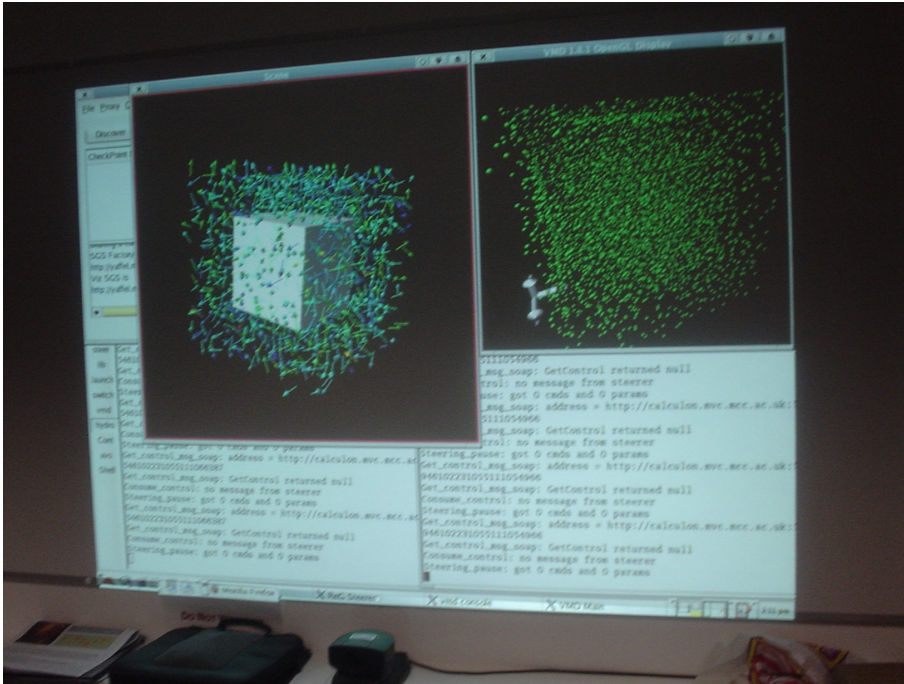


FIG. 4: A picture of the output from a coupled model running on a prototype system consisting of a box of argon atoms. On the left is the output from a visualization connected to the CFD component and on the right is another from the MD component. The visualization applications are themselves separate components in the architecture.

in figure 3, this can lead to problems when both the steering client and any one of the application components attempts to send a message simultaneously. Ultimately, a deadlock will occur when one SGS attempts to call another SGS that must itself call the first. We deal with this situation by allowing the component-initiated communication to time-out and then try again.

III. APPLICATION TO HYBRID MOLECULAR DYNAMICS

We demonstrate the use of steering coupled models with the RealityGrid library for a two component model coupling MD and hydrodynamics. We are able to change several global and non-global parameters and observe the collective results by monitoring the molecular system via online visualization using the RealityGrid modified version of VMD (figure 4). We use an AVS/Express-based visualization to monitor the continuum region. The system is set-up with a small MD region surrounded by a large continuum domain. The amplitude and frequency of pressure wave oscillations coming from the continuum hydrodynamics region are freely changed and steered in the molecular region.

Over the past few years, we have been developing a new approach to the modelling and simulation of complex fluid flow for application to a wide range of problems in soft condensed matter science and engineering. Our approach is based on a multiscale hybrid scheme, in which two or more contiguous

sub-domains are dynamically coupled together, the simplest case corresponding to one sub-domain being described by molecular dynamics, the other by continuum fluid dynamics (CFD) [5–7]. The motivation for this hybrid approach to the description of fluids is that it enables us to tackle problems that are not addressable by any one single technique; such problems include situations in which hydrodynamic effects influence molecular interactions at interfaces, lipid bilayer membranes and within individual macromolecules or assemblies of them. Since the computational overhead of coupling the molecular and continuum regions is very low, we can use this approach to perform coupled simulations using small MD domains which are free of finite size effects, whilst reducing the computational cost compared with fully atomistic simulations. The approach can be used to probe limitations to CFD descriptions which are known to break down in situations where molecular phenomena play a large role, such as the moving contact line between two immiscible fluids and a boundary.

A. Model structure

We have developed a hybrid molecular dynamics scheme using two, independent simulation codes implementing molecular dynamics (MD) and a finite volume discretisation of the equations of fluctuating hydrodynamics (Hydro) [8]. These two models are heterogeneous from a physical point of view but are also very different programmatically because MD is a particle code, while Hydro is mesh

based. These considerations and the need to have flexible/interchangeable components lead us to create a coupled architecture with two independent programs which exchange data rather than a single, monolithic code.

This component-based approach has significant advantages. Not least of these is that each individual component can be developed, tested and used independently as a standalone code. In practise, each component can be run on a different machine and it is easy to extend this scheme to multiple components (three, four, *etc.*) as is required when adding an extra level of description to the physical model (*e.g.* quantum mechanical).

The architectural diagram illustrating how the two application components interact is shown in figure 5. The main program is modified in order to accommodate a call to a subroutine from the ‘hybrid’ library which takes care of handling, preparing and communicating the data required by the hybrid coupling. This ‘hybrid’ code is rather specific and needs access to the data structures of the program (hydrodynamics or MD). In our case we have hidden the code behind a simple interface which provides the functionality needed. As a result, the ‘hybrid’ code has to be written only once and another molecular dynamics code (*e.g.* a parallel code) could easily be deployed within the hybrid MD coupled model by providing the same high-level interface.

B. Component synchronisation and data exchange

A key issue for coupled models is the synchronisation of the various components. This is typically determined by the nature of the information which the components must exchange with one another and the stage during the calculation cycle at which this exchange occurs. In the models we are dealing with, each component tackles a separate physical region of the system and the data exchanged between them provides the boundary conditions for each region. The frequency of this exchange is model dependent, *e.g.* the timestep of a molecular dynamics (MD) simulation might be one hundredth of that of an associated continuum simulation. In this case, the two components might exchange data after every step of the continuum simulation (and thus every 100 steps of the MD simulation). For the models we consider here, each component is assumed to be at the same point in (simulated) time whenever a data exchange takes place.

Although several specialised coupling frameworks exist, particularly in the field of atmospheric and ocean modelling [9–12], we have chosen to use a very simple solution since our current models contain only two components and do not use any particular features of such frameworks. To enforce synchronisation and permit data exchange, we have constructed a communication “switch” service that

implements a store and forward message-passing scheme. The interface to this switch service consists simply of a *put* operation and an blocking *get* operation. It is the use of the latter that enforces inter-component synchronisation. In order to avoid deadlock in such a scheme, each component executes its ‘puts’ before its ‘gets’ [13].

Our experience within the RealityGrid project has consistently shown that establishing some sort of socket-based connection between applications running on different machines is plagued with problems — even on machines explicitly stated to be ‘Grid-enabled’. In the worst case, the nodes on which a job runs might not have any internet connectivity. More common is the case where Network Address Translation or tunnelling (via a head node) is used so that ‘backend’ nodes can connect out to the internet but do not themselves have unique IP addresses, thus preventing incoming connections to software running on these nodes, even if local firewall rules permit.

Components connect to and communicate with a “switch” service, creating a communications network with a star topology. Each component is assigned a unique identifier which is used for message routing purposes.

Although indirecting messages through the switch imposes a latency penalty, there are several compensating benefits:

- Loose coupling of components. Because messages are indirected, an individual component need not know the details of location of its peers.
- Transience of components. Because the switch buffers undelivered messages, components may be disconnected and reconnected without destabilising the coupled application. It should be noted that this may impose a performance penalty on any peer components performing a blocking wait for messages from the disconnected component. When combined with performance monitoring information, the ability to seamlessly disconnect components facilitates full performance control through migration of components across resources.
- Firewall-friendly. This communication scheme is firewall friendly because any connection is always initiated by the model component and made to the switch. Provided the switch is located on a resource with a suitable incoming-connection firewall policy, individual components located on firewalled machines may still intercommunicate.
- Diagnostics. The switch provides a convenient, single point at which any problems with the communications may be readily investigated.

Message sending is non-blocking, with the switch buffering undelivered messages. A component may

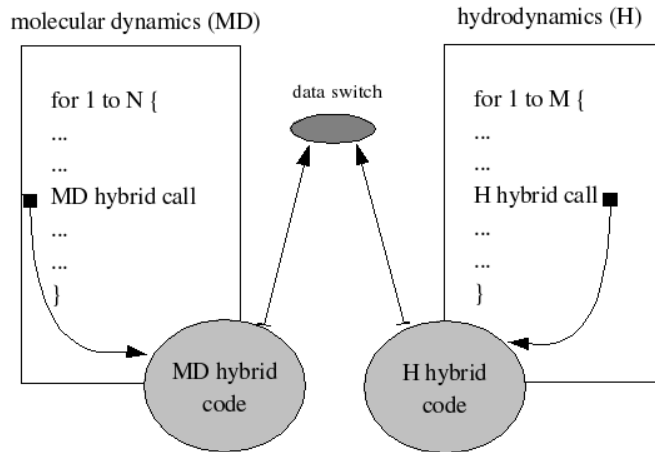


FIG. 5: The implementation of the hybrid molecular dynamics code requires minimum perturbation of the original molecular dynamics and hydrodynamics codes. The exchange and processing of information to transform between physical descriptions is performed by opening a channel in the main loop of each of the codes by calling a subroutine in our ‘hybrid’ library. Each simulation code performs a standard calculation; only the data being used is affected by the presence of the other component.

perform either blocking or non-blocking message receipt. These semantics facilitate component synchronisation.

IV. CONCLUSIONS AND FURTHER WORK

This paper signals the beginning, but only the beginning, of a journey into the world of steered coupled models which can be deployed on a general purpose Grid. It is straightforward to extend the scheme described here to the case of multiple components; within computational materials science and systems biology [14] there is a need to couple levels of description from the quantum mechanical, through classical molecular dynamics and coarse-grained particulate representations to the continuum level described by computational fluid dynamics, finite-element descriptions and so on. Both the EPSRC-funded Integrative Biology e-Science Pilot Project and the BBSRC-funded Integrative Biological Simulation Bioinformatics and e-Science Programme Project have an interest in such multi-level coupled modelling capabilities.

Steering such multicomponent coupled models is

not expected to provide any fundamental problems given that the couplings are always between nearest neighbours in the space-time hierarchy of model descriptions. Moreover, we are particularly interested in developing an ability to insert and delete component models on-the-fly when highest or lowest levels of resolution are warranted. Grid deployment is likely to be important for obtaining good load-balancing of such coupled simulations and we expect to be able to use existing and developing RealityGrid middleware to facilitate this. The problem of performance control is particularly interesting for coupled models; we expect to be able to provide this initially via manual migration of individual components between compute resources on a Grid. Ultimately, we hope to include automated performance control within the RealityGrid steering system, but that is another story.

Acknowledgments

This research was supported by the EPSRC RealityGrid project GR/R67699 and the EPSRC Integrative Biology project GR/S72023.

-
- [1] C. F. Sanz-Navarro, S. D. Kenny, A. R. Porter, and S. M. Pickles. Real-time visualization and computational steering of molecular dynamics simulations of materials science. In *paper presented at the UK e-Science All Hands Conference, Nottingham, 2004*. Available from <http://www.allhands.org.uk/2004/proceedings/papers/107.pdf>.
- [2] S. M. Pickles, R. Haines, R. L. Pinning, and A. R.

- Porter. A practical toolkit for computational steering. *Phil. Trans. R. Soc. Lond.*, A363, 2005. In press.
- [3] J. Chin, J. Harting, S. Jha, P. V. Coveney, A. R. Porter, and S. M. Pickles. Steering in computational science: mesoscale modelling and simulation. *Contemporary Physics*, 44(2):18, 2003.
- [4] The steering library and documentation are

- available from: <http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid>.
- [5] R. Delgado-Buscalioni and P. V. Coveney. Continuum-particle hybrid coupling for mass, momentum, and energy transfers in unsteady fluid flow. *Phys. Rev.*, 67:046704, 2003.
- [6] R. Delgado-Buscalioni, E. G. Flekkøy, and P. V. Coveney. Fluctuations and continuity in particle-continuum hybrid simulations of unsteady flows based on flux-exchange. *Europhys. Lett.*, 69:959, 2005.
- [7] R. Delgado-Buscalioni, P. V. Coveney, G. D. Riley, and R. W. Ford. Hybrid molecular-continuum fluid models: implementation within a general coupling framework. *Phil. Trans. R. Soc. Lond.*, A363, 2005. In press.
- [8] G. De Fabritiis, R. Delgado-Buscalioni, M. Serano, and P. V. Coveney. 2005. preprint.
- [9] W. D. Collins, C. M. Bitz, M. L. Blackmon, G. B. Bonan, C. S. Bretherton, J. A. Carton, P. Chang, S. C. Doney, J. J. Hack, T. B. Henderson, J. T. Kiehl, W. G. Large, D. S. McKenna, B. D. Santer, and R. D. Smith. The Community Climate System Model: CCSM3. *Journal of Climate*, 11(6), 1998.
- [10] C. Goodrich, J. Lyon, and T. Guild. Coupling solar terrestrial models using overture and intercomm. In *Proceedings of ISSS-7, 26-31 March, 2005*, 2005.
- [11] R. W. Ford, G. D. Riley, M. K. Bane, C. W. Armstrong, and T. L. Freeman. GCF: A general coupling framework. *Concurrency and Computation: Practice and Experience*. To appear.
- [12] *Program for Integrated Earth System Modeling*. <http://prism.enes.org/>.
- [13] R. W. Ford and G. D. Riley. *The Met Office preFLUME Project: Single Model Software Architecture, v1.2, Manchester Informatics Ltd., The University of Manchester, 2003*. Crown Copyright 2003, published on the Met Office web site: <http://www.metoffice.com/research/interproj/flume>.
- [14] P. V. Coveney and P. W. Fowler. Modelling biological complexity: a physical scientist's perspective. *J. R. Soc. Interface*, 2(4), 2005. Available on-line at <http://www.pubs.royalsoc.ac.uk/interface.shtml>, DOI: 10.1098/rsif.2005.0045.